$2.50

64

STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS
PRIMES
STRINGS

Popular Computing

The magazine for those interested in the art of computing

July 1978 Volume 6 Number 7

5651		1146791
	(2)	
5653		1146793
	(4)	
5657		1146797
	(2)	
5659		1146799
	(10)	
5669		1146809
	(14)	
5683		1146823
	(6)	
5689		1146829
	(4)	
5693		1146833
	(8)	
5701		1146841

Patterns in Primes

Patterns in Primes

The two strings of consecutive prime numbers shown on the cover have the same pattern of differences (as given by the numbers in circles). Thus:

5653 - 5651 = 1146793 - 1146791

and so on for the other seven differences.

These are the longest strings of consecutive primes that we can find with the same difference pattern.

We can find other repeating patterns, though, such as:

13901	21557	28277
13903	21559	28279
13907	21563	28283
13913	21569	28289
13921	21577	28297
13931	21587	28307

with the difference pattern 2-4-6-8-10. Also:

21577	342037
21587	342047
21589	342049
21599	342059
21601	342061
21611	342071
21613	342073

are two sets with the pattern 10-2-10-2-10-2.

Publisher: Audrey Gruenberger
Editor: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald
Contributing Editors: Richard Andree
William C. McGee
Thomas R. Parkin
Advertising Manager: Ken W. Sim
Art Director: John G. Scott
Business Manager: Ben Moore

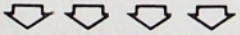
POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each.

We cannot conclude that any pattern of differences will repeat indefinitely (if we could, then we could conclude that there were infinitely many twin primes), but since the differences between consecutive primes are generally small (one must go to 79 167 733 in the natural numbers before first reaching a difference of 184) it follows that most difference patterns will repeat. For example, the difference pattern exemplified by the following strings of four consecutive primes:

101	13001	101111	104386151
103	13003	101113	104386153
107	13007	101117	104386157
109	13009	101119	104386159

occurs 4917 times among the first 6,000,000 primes.

Of all the difference patterns that repeat, there must be a repeating pattern that is longer than any other. We are hardly likely ever to find that pattern. We could expect, however, to find a repeating pattern that is longer than 9 primes.



So our 17th contest is very simple: our customary prize of a Texas Instruments TI-58 calculator will go to the person who produces the longest repeating difference pattern in consecutive prime numbers, and whose entry arrives by October 15, 1978 at:

Contest 17
POPULAR COMPUTING
Box 272
Calabasas, California 91302

There will be only one prize. What counts in our contests is the quality of computing, including well-written computer code, adequate documentation, discussion of the method of attack on the problem, and the like.

A Personal Account

Fred Gruenberger

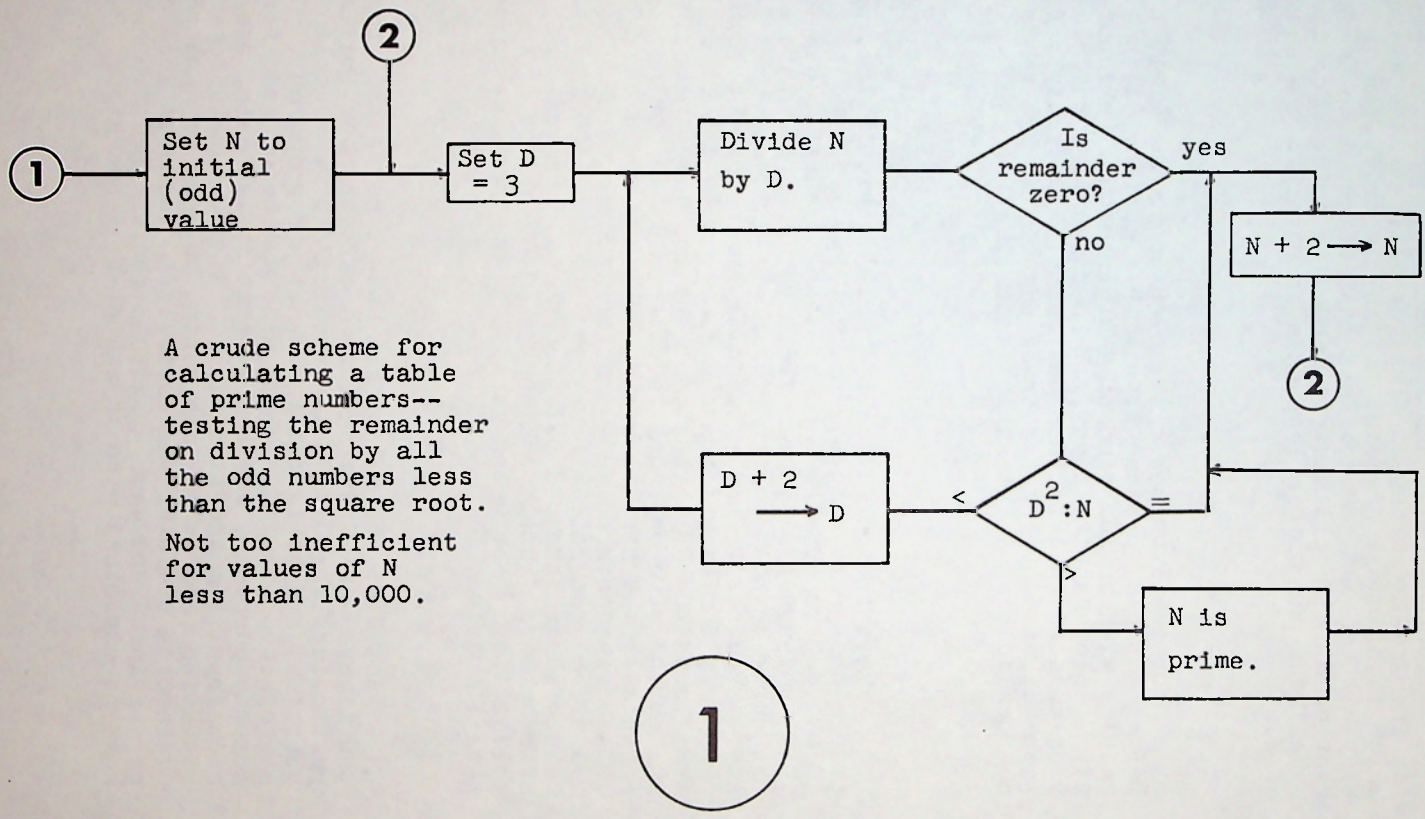
Some people collect stamps; others collect matchbook covers; there are those who collect buttons.

I started collecting prime numbers in 1949, when the University of Wisconsin Computing Center acquired an IBM 605. The accompanying flowchart (1) shows a simple algorithm for sifting out a table of primes. It is simple enough to fit into the 80 wired program steps of a 605. That machine could execute 2000 add-type steps per second (but was abysmally slow on multiplication or division) and could be looped; that is, one card could be stalled in the feed chain while endless calculation took place. Thus, although algorithm (1) is crude and very inefficient (and no one, at that time, had shown me a better algorithm) it did fit the available machine and in the range around N = 10,000,000 it could produce pretty primes at the rate of about 100 per hour.

There was a feature to the 605 calculation that was unique. The value of D, the variable divisor, could be observed in the neon display lights, so that it was possible to watch a new prime as it developed, so to speak. In any range (the 605 program would function for values of N up to 13 digits, which was 100,000 times as large as the upper limit of the best previous table), for example, one could see twin primes as they emerged, and one could be led to believe devoutly that twins would always occur. We seldom get an opportunity any more to watch numbers performing for us.

There had been many printed lists of prime numbers prior to this time, the most notable being D. N. Lehmer's List of Prime Numbers from 1 to 10,006,721, Carnegie Institute, Washington, Pub. 165 (1914)--also available in a Dover reprint.

Look at that date!--all the 7-digit primes sifted out at a time when the only mechanical aid obtainable was a key-driven adding machine. The Introduction to Lehmer's table is still the best discussion of the history of prime numbers.



A crude scheme for calculating a table of prime numbers--testing the remainder on division by all the odd numbers less than the square root.

Not too inefficient for values of N less than 10,000.

Algorithm (1), when implemented on a computer, is far from satisfactory. First of all, it uses division, which is a costly process. Then, it calls for dividing N by all the odd numbers less than or equal to the square root of N, when all that is needed is a test for divisibility by all the odd primes less than or equal to the square root of N.

For small values of N, even up to the limit of Lehmer's table, one can live with algorithm (1). On a machine with an add time of 4 microseconds (which is a rather slow machine today), Lehmer's entire table could be recalculated in less than one shift. On today's fastest machine (still using the crude algorithm) the table could be recalculated in about 15 minutes.

There seemed little point to recalculating the primes that were already in print, so I jumped to the 40,000,000 range when I had access to a real computer in 1954. I wrote a program for algorithm (1)--this was about the upper bound to my programming ability at that time--and talked our friendly customer engineers into running it every day during maintenance time on the IBM 702. In those days the CE's took over the complete installation for an hour or more each day to clean all the moving parts, especially the tape servos. The CPU would usually be idle during this period, and the program used only the CPU; all primes generated in an hour--perhaps 1000 or so--could be stored and punched out when normal work resumed. Thus, on most days I could add another 1000 primes or so to my collection. Not being greedy, I was quite satisfied with this situation.

But Clarence Poland was not. He noticed that the CPU lights were flashing during maintenance time (and, in particular, the DIVIDE light seemed to be stuck in the ON position) and he looked into what was going on. It did not disturb him that the machine was being used for non-company purposes, but rather that it was being used with such appalling inefficiency. He set out to remedy that situation (and it is interesting to keep in mind that Clarence's training was entirely in business applications of machines) and came up with algorithm (2), which can best be explained with an example. See Figure 3.

For primes in the range from
503 to a maximum of 529

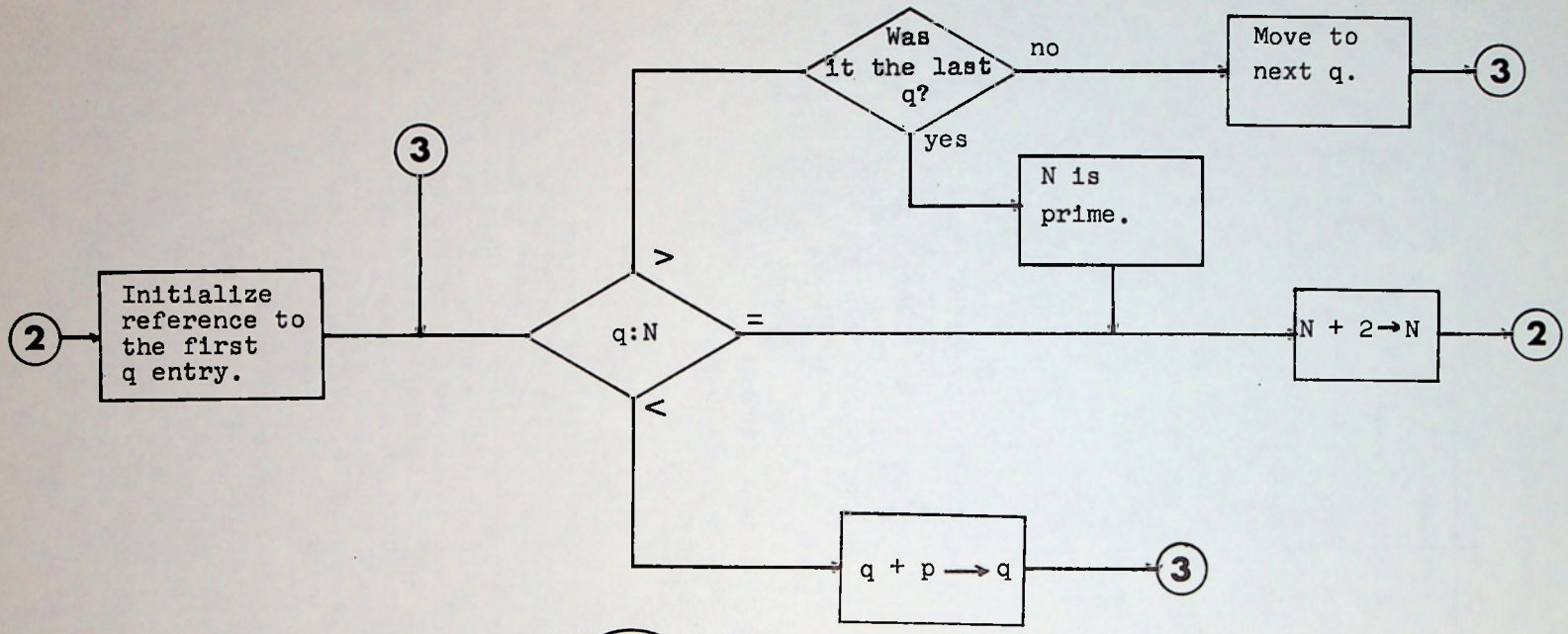
p	q
3	501
5	500
7	497
11	495
13	494
17	493
19	494
23	483

We construct, during the housekeeping phase of the program, a table in storage. The argument of the table is a list of the odd primes up to the square root of our impending range. For the indicated range in the example (503 to 529) this would be the primes from 3 through 23. For each such prime, p, the functional value, q, is the nearest multiple of p less than or equal to the initial value of N. Thus, if N is to start at 503, each q entry in the table is formed by calculating:

$$\left[\frac{503}{p}\right] \cdot p$$

The square bracket operation ("greatest integer in") is precisely how any computer does integer arithmetic, so the calculation of table (3) consists of one division and one multiplication for each p value. The p values themselves could be read in, or be generated by using algorithm (1), or be generated by a lower-level use of the procedure being described here. It makes little difference; the work of calculating table (3) is done only once for each run on the machine. Even when this table is long (e.g., 800 entries when the range on N is around 40,000,000) its construction consumes only a few seconds of CPU time.

After the table has been constructed, then the logic shown as algorithm (2) begins. The table of q values simply lists all possible multiples of anything possible, less than or equal to the number, N, being tested. We compare each q value in turn to N. If equal, then N cannot be prime (since it is then a multiple of some smaller number). If q is less than N, then q is updated (by addition) by its corresponding p, and we compare again. And if q is greater than N, then we move to the next q, or, if it was the last q, then N is prime.



2

An improved algorithm for sifting a table of consecutive prime numbers, originally devised by Clarence B. Poland III.

After the initial set of divisions, done only once, the new algorithm uses only addition and some straightforward address modification to move systematically through the p/q table.

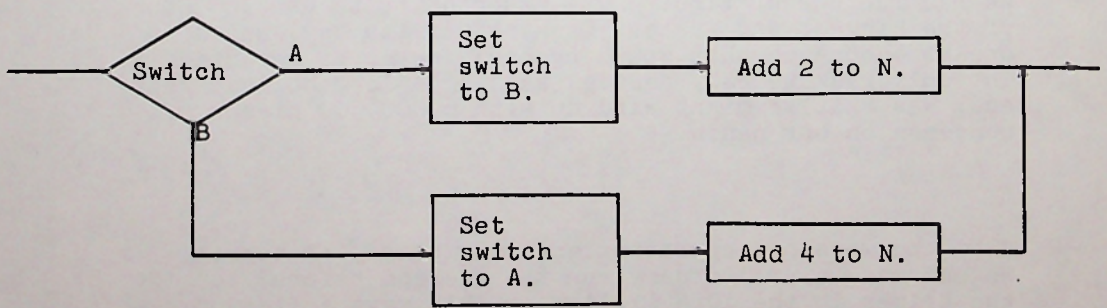
The Poland algorithm on the same machine (the 702) produced a gain in speed by a factor of 4; that is, an hour's use of the machine would now net me perhaps 4000 primes in the 40,000,000 range.

At this point in my personal history, I changed jobs, to join the RAND Corporation. My new employer used a binary computer, which was not so commonplace then. My own evolution had been through calculators (the 602A, 604, and 605) and decimal computers (the 650 and 702) and the shift to binary was a big step. (It is now difficult to believe that up to 1962 or so, over half the computers in the world were decimal.)

It seemed like a good idea to get familiar with binary logic by re-doing a familiar problem; namely, the primes algorithm. Charles L. Baker, who had been volunteered to indoctrinate me into the mysteries of binary computing, examined the Poland scheme and added some improvements.

For example, the flowchart (2) calls for updating each q value by adding to it its corresponding p value. Table (3) shows that this updating will always result in an <u>even</u> value for the new q, and only odd values could be of <u>any</u> use. Therefore, the change to $q + 2p \longrightarrow q$ must improve the speed of the algorithm.

Similarly, algorithm (2) as shown calls for examining only odd values of N; the even values are skipped by adding 2 to each trial N, with the initial N being odd. The same scheme can be extended:



to skip over all multiples of 3 also.

Baker also arranged for extremely tight code for the IBM 704, using index registers for movement through the p/q table, and making efficient use of the 704's CAS (Compare Accumulator to Storage) operation. His final code spent 90% of its time in a 2-instruction loop. Some preliminary runs showed that, in the 40,000,000 range, the modified algorithm on the 704 would yield around 50,000 primes per hour.

It struck us that we had something good going here and that, given some machine time, we could corner the market in prime numbers. This was 1957, and some available machine time appeared: the July 4 weekend in 1957 was four days long, and the company intended to turn off the 704 all four days. We proposed that the machine be turned over to us, to be used to calculate a very large table of primes. At that time, the 704 rented for around $400 per hour, so we were proposing using about $40,000 worth of machine time for nothing.

Our boss broached this matter with the local IBM manager who did the only thing possible: he passed the buck to IBM's World Headquarters. IBM did grant us permission to use the machine, but the permission arrived in November of 1957.

The July 4 weekend wouldn't wait, however, so our boss stuck his neck out and told us to go ahead. We did; Charlie and I babysat the machine for some 110 hours that weekend, winding up on Monday morning with 6,000,000 primes.

At least we hoped so. I am a firm advocate of the principle that no program should be committed to production without being thoroughly tested. In this case, we were stuck. We had no clear-cut way to test the program, and we had run out of time. The opportunity to use 100 hours of the biggest and fastest machine (of its day) was a one-time bonanza that would probably never be repeated. The only risk we ran, besides wasting a long holiday weekend, was that we might wind up with 6,000,000 pieces of nonsense on our hands.

We were not operating entirely blind, of course. We had made a preliminary run the weekend before, to sift the primes in the 10th million. This gave a list of 62,090 primes, which agreed with the published Meissel counts.

A note about that. Meissel derived a formula (see Uspensky and Heaslet's Elementary Number Theory) which works essentially like this: if the independent variable is given a value of a prime, the formula furnishes the index of that prime; that is, it tells which prime it is. Uspensky and Heaslet illustrate the use of the formula with the value N = 7993; the calculations fill two pages of their book and give the result 1007. It would take very little more computing effort simply to apply algorithm (1) and produce those other 1006 primes. The work of evaluating Meissel's formula increases with the size of the input variable. "By means of this formula, but after laborious calculation, Meissel found that below 100,000,000 there are 5,761,460 primes." Unfortunately, after all that work, Meissel seemed to off by 5, which is quite a serious error.

It seems clear that Meissel's formula is novel and interesting, but impractical. For large numbers, the work of evaluating the formula is formidable.

In any event, the preliminary run had given us some confidence in our program. On the other hand, on July 9 we had 6,000,000 numbers, and a gnawing fear that there might be a flaw in our reasoning and that the list of "primes" might be full of errors. The question was largely academic, of course, because we had recorded the results (by punching) in the form of differences between the primes, and in binary, so we weren't even equipped to examine our results too carefully.

We could brag, however, and proceeded to do so. We wrote Prof. D. H. Lehmer (D. N. Lehmer's son) with the news that we had 6,000,000 of something, and the last of them was 104,395,289. There followed a period of six weeks of silence.

Then we got a letter from Prof. Lehmer--a quite remarkable letter. It was written in his usual courteous and scholarly style, so I will paraphrase it slightly.

First he said "Congratulations." This is a much larger compliment than it looks. Prof. Lehmer had been working on this same project for some time, utilizing the idle time on the machine at Berkeley. He was up to 37,000,000 in the natural numbers, and we had just wiped out his favorite piece of research.

Next he said, of the number 104,395,289, "That's right." He had programmed Meissel's formula, fed it the number 104,395,289, and got back the number 6,000,000. Thus, he had greatly improved the chances that our table was correct. If our program had contained any systematic error, it would be miraculous that we would stumble exactly right on the 6,000,000th prime.

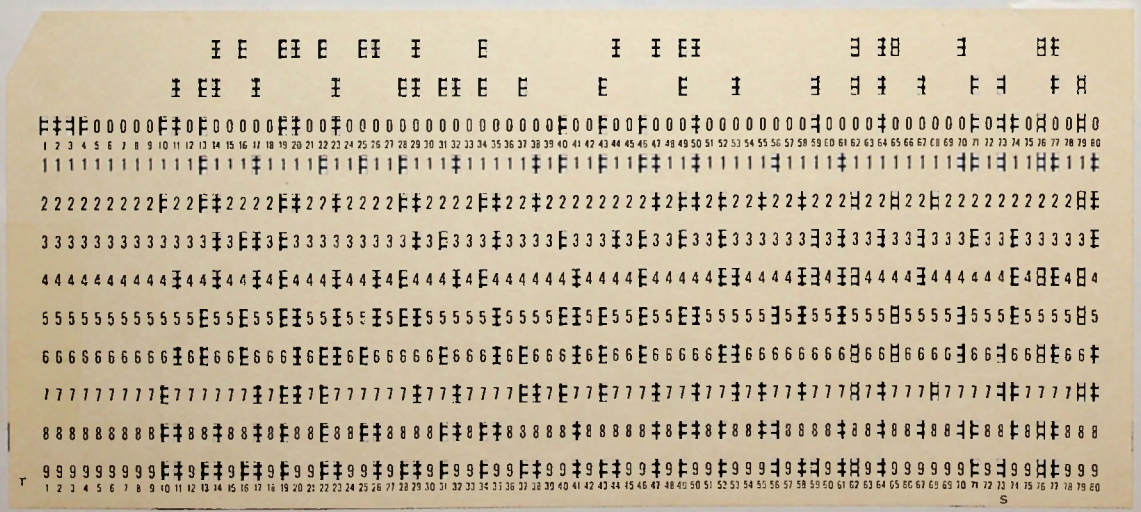
Finally, Prof. Lehmer said, in effect, "You might be interested in how I was going about it. Here is a card from my table."

An exact copy of the card he sent us is attached to this issue. It took us some time (since one would rather die than ask silly questions) to deduce from this card just what method Lehmer had been using, and how much better it was than the Poland-Baker algorithm.

I do not intend to spoil your fun. You already have more information than we had then, so you should be able to deduce from the card what Prof. Lehmer was trying to tell us. You are allowed one more clue: the card is a core dump, and the card punching machine that made it feeds its cards 9-edge first.

On the other hand, perhaps it was just as well that we hadn't known of a better algorithm. Lehmer's method, it turns out, runs about 60 times as fast as algorithm (2). If we had known of it, we might have used it, and wound up with 360,000,000 primes.

As it was, the 6,000,000 primes we had were something of an embarrassment at the time. When we eventually printed them out, they filled 4800 11x14 tab sheets, and that large a table would be difficult to get published.



Fortunately, microcards were invented just then, and the table appeared as the first book published on those cards. The book The First Six Million Prime Numbers consists of 62 3x5 cards, with each card bearing 38 reproductions of the tab sheets. It includes eight pages of text describing the method of calculation in more detail. The book is also available in microfiche.

The publisher asked us to estimate the optimum print run for this book, and Charlie and I figured that about 35 copies would saturate the world, including the two copies needed to secure the copyright, and a copy for each of us. We sent IBM a copy, too--it was their machine that we used. The last time I checked, some 800 copies of the book have been sold, many of them to individuals.

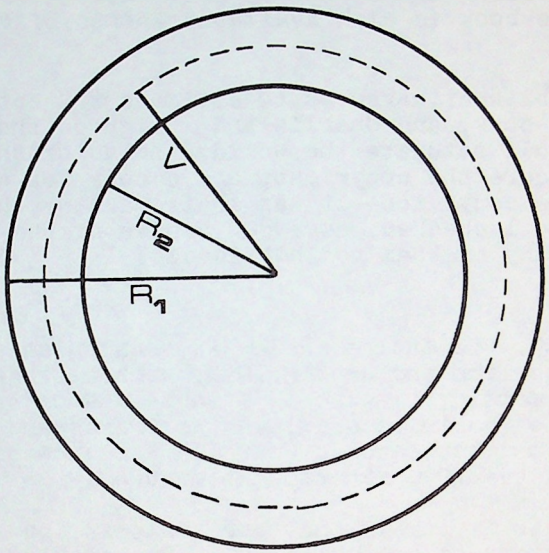
In 1959, the entire table was recalculated, using Lehmer's algorithm and an IBM 7090, with a program written by George Armerding. The 1957 table was on tape, so all that was done was to recalculate the primes and compare. There were no discrepancies (our 1957 program was finally tested!) and the 7090 run took 21 minutes.

Thus, it is likely that our table is the last table of prime numbers to be published. On today's best machines, it is possible to sift out primes (in the 9-digit range) at the rate of a million a minute, so there would be little point to the use of a printed table. Whatever it is that someone might want to do with primes, he can calculate the ones he needs faster than he could look them up.

The original 4800-page table was bound, and is in the RAND Corporation library. The Microcard reproduction is sold by the Microcard Corporation, a division of NCR.

The algorithm that Prof. Lehmer used was described very briefly in an article in our issue No. 13. We will elaborate on it in a subsequent issue.

THE ROUND STRIP



The large circle shown in the Figure has an area of 10,000 square units; its radius is R_1. The circle with radius R_2 has an area of 9900 square units; that is, 100 square units have been stripped away from the large circle. The "length" of the annulus between them is given by 2π times the average radius, $(R_1 + R_2)/2$.

The process is to continue, stripping off 99 square units for the next circle, then 98 square units,..., until finally one square unit is removed by the 100th circle. The Problem, then, is: What is the total length of all the 100 annuli? (We make it to be 28712.423 units.)

We have these test numbers:

R_1 = 56.4189583547

R_2 = 56.1361547776

average = 56.2775565662

first length = 353.6023165

ART OF COMPUTING 14

The Only Real Counters

Man has devised many counting schemes. We count some things by dozens, others by 20's. Book pages are counted by 32's; card suits by 13's; days by 365's, or 28's, or 29's, or 30's, or 31's.

Consider a church clock, which is a counting device. The big hand counts by 60's, in an endless cycle. The little hand counts by 12's. Each of these counters has its limit, and when the limit is exceeded, the device simply starts counting again. We seldom refer to a time as 217 minutes past 8 o'clock; we prefer to call it 37 minutes past 11 o'clock.

Only mathematicians can contemplate--and make use of--counters that function to infinity. All real (physical) counting devices, being finite, count to some limit, and then roll back to zero and count over. The limit is called a modulus. We are so accustomed to counting in modular arithmetic that we are usually unaware of it.

For example, today, let us say, is Monday. Seven days from today is a Monday, as is 70 days from today, or 700 days before today, or 70,000 days ago--and in the last case it would take some figuring to determine what year it was. But since we count days of the week by 7's (and have, for all of recorded history), we are assured that each of the days referred to can be labelled Monday. All the numbers involved here are divisible by seven. We can express that fact in a notation that looks formidable:

$$700 \equiv 0 \mod 7$$

which can be read in its technical form:

"700 is congruent to zero modulo 7,"

or in simple English:

"700 yields a remainder of zero on division by 7."

The new notation shows that:

$$364 \equiv 0 \mod 7$$

which immediately extends to:

$$365 \equiv 1 \mod 7.$$

Thus, if today is a Monday, then 364 days from now will be a Monday, and 365 days from now will be a Monday-plus-one, or Tuesday. Today's date a year from now moves one day in the weekly count (unless, of course, a leap day intervenes).

For another example of this kind of counting, consider the game of Oware (described in issue No. 55). The rule of play in Oware calls for distributing markers around the 12-cell board, with no markers to be returned to their source cell. Thus, each play operates on the number of markers modulo 11. If the number of markers to be distributed is 11, or 22, or 33 (that is, congruent to zero mod 11), there will be complete integral sweeps around the board. For 12 markers, there would be a complete sweep plus one cell, and so on. Each player can readily predict where he will land with any given stack of markers simply by calculating the remainder on division by 11.

The new notation seems to make some things more complicated, such as when we write:

$$3474428667 \equiv 1 \mod 2$$

instead of simply saying that 3474428667 is odd. However, it is a standard trick of mathematicians to devise a new (and invariably very compact) notation and then proceed to explore that notation as an entity by itself. For example, matrix notation (a matrix being only a rectangular array of numbers) has been developed into a powerful tool, with an entire algebra of its own. So it is with congruence notation.

The manipulation of congruences is not the point here, however (see any number theory text for the rules of operating on congruences). What is the point is that congruence notation and modular arithmetic are basic concepts to computing, inasmuch as the heart of computers is counters and counting, and all counting done in physical entities is modular.

This basic idea also leads us directly to a device for forming scrambled sequences of numbers. When we write:

$$\mod 13$$

we have momentarily formed a number system that has only 13 elements all told; namely, the positive integers from zero to 12. Another number, like 29, is equivalent to 3 (that is, the remainder on division by 13).

Let us form a table of powers of 2, with each element in the table expressed mod 13:

2	We start with the first power of 2.
4	Form each new number by doubling.
8	So far everything looks normal.
3	But now we have $16 \equiv 3 \mod 13$.
6	We are still doubling. Or, in normal notation, we would be up to 32 here, but $32 \equiv 6 \mod 13$ also.
12	
11	
9	
5	
10	
7	
1	And the sequence would continue 2, 4, 8,...

The process could continue indefinitely, but we have completed a full cycle. We have obtained all the possible non-zero values, mod 13, in somewhat scrambled order.

Try the same scheme again, but mod 17. This time we do not get all possible non-zero numbers, but only 8 of them. For mod 19, we will get 18 (that is, every one possible), but for mod 23 we will get only 11. For mod 31 (notice that we are considering only prime moduli) there will be only 5. We can say with certainty that for a prime modulus, p, the number of elements, a, in a complete cycle will always be given by:

$$a \mid (p-1)$$

(a divides p-1) which is to say that the cycle length will always be an integral sub-multiple of (p-1). Thus, if we follow the same procedure as before, but using mod 47, we know before we begin that the number of terms in one cycle will be one of 46, 23, 2, or 1--those are the only possible values. For mod 97, we have these possibilities: 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, or 96 (the cycle length is actually 48).

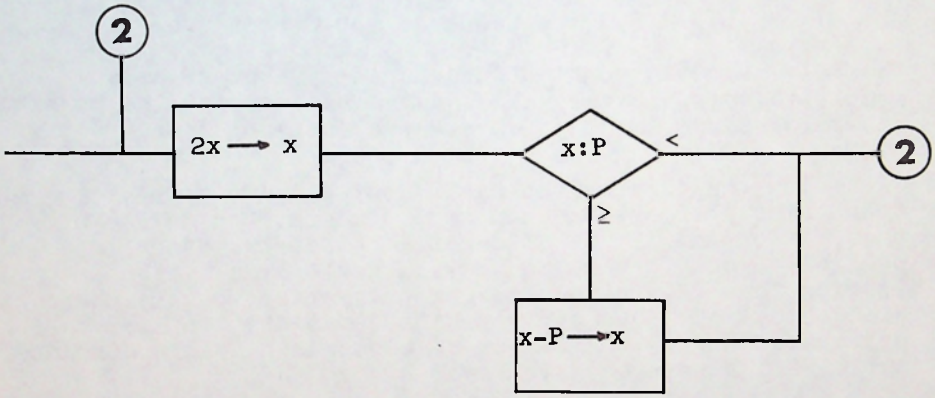
When the modulus gets large, the calculations become long, tedious, and surprising. Thus, for mod 100013747, the table of powers of 2 goes all the way; the cycle length is 100013746, and the simple procedure produces a list of 100013746 8-digit numbers in scrambled order. On the other hand, for mod 100013761, the list would be only 9471 numbers long in a complete cycle.

Some moduli lead to full-sized cycle lengths, and some don't. About 35% of all prime moduli seem to be the kind that do (another 30% lead to half-sized cycle lengths). Just about the only way to tell which is which is to perform the calculation, although there are some rules and short-cuts. For example, the only numbers that are eligible to produce full-sized cycles are primes of the form 8k+3 or 8k+5. Thus, 97 is not even eligible to produce a sequence that is 96 numbers long, but 43 is eligible. The rule is helpful but not an algorithm. Thus, it tells us that both 107 and 109 are eligible to produce full-sized cycles; 107 does, and 109 does not.

The scrambling effect achieved by our powers of 2 scheme is not too good. We have been implementing this recursion:

$$x_{n+1} \equiv 2x_n \mod P$$

for which the flowchart logic is:



The scrambling effect can be improved by using a larger multiplier than 2; for example:

$$x_{n+1} \equiv 893871739\ x_n \mod 100013747$$

β

where the multiplier has been selected as the 7th power of 19; that is, a prime to a prime power, less than the modulus.

This gives us a pretty fair random number generator. The recursion above will produce all the numbers between 1 and 100013746 in a scrambled order.

But this introduces a new problem. The operation "mod 100013747" involves division, which is always costly in terms of CPU time. If we could take the recursion:

$$x_{n+1} \equiv M\, x_n \bmod P$$

and, instead of using a prime for the modulus, use instead a value like 2^{32} (for a 32-bit word size machine), then the modulus operation reduces to retaining only the right half of a multiplication. Most packaged random number generators (e.g., those buried in BASIC interpreters) are of this type, heavily dependent on machine characteristics. The writers of such generators must then select a value for M carefully, and the proper choice is not always obvious (and the writers of those interpreters are not always competent number theorists). However, the "prime to a prime power" notion usually suffices.

If the requirements imposed on the output of a random number generator are not too stringent, then a generator like the one shown at (β) is usually quite satisfactory. We can increase the cycle length (and usually improve the output) by increasing the modulus. Ultimately, by compounding two or more independent generators, we can make the cycle length arbitrarily long and the output arbitrarily good (that is, such that it will pass the standard tests of randomness). However, users should be aware that the writers of most packaged generators are singularly uninterested in either long cycle length or quality output; their only concern is that of minimizing CPU time. As with any program that is handed to you, you are free to test it as much as you please.

There are eight standard statistical tests for quality of randomness. They were briefly described in our issue No. 33 (December, 1975). In a subsequent essay, several of them will be discussed at greater length.

PC64-20

Our once-a-year inventory clearance sale is on again.

Complete Your File of *Popular Computing*

On orders received up to July 31, 1978:

Issues 1 through 38, if available, 50¢ each.

Issues 39 through 49, $1.00 each.

20% discount on orders totalling $12 or more.

Plus: During this sale period, 2-year renewals will be accepted for $29 (U.S.), or $35 (all other countries), regardless of when your current subscription expires.

Most Back Issues Are Still Available:

JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC		
			1	2	3	4	5	X	X	X	9	Vol. 1	1973
10	11	12	X	14	X	X	17	X	19	X	21	Vol. 2	1974
22	23	24	25	26	27	28	29	30	31	32	33	Vol. 3	1975
X	35	36	37	38	39	40	41	42	43	44	45	Vol. 4	1976
46	47	48	49	50	51	52	53	54	55	56	57	Vol. 5	1977
58	59	60	61	62	(63)	64	65	66	67	68	69	Vol. 6	1978